IN THE UNITED STATES DISTRICT COURT
FOR THE EASTERN DISTRICT OF TEXAS
MARSHALL DIVISION

IMPLICIT, LLC, Plaintiff, v.	§ § § § §	
IMPERVA, INC.	§ § §	Case No. 2:19-cv-00040-JRG-RSP LEAD CASE
FORTINET, INC.	§ § §	Case No. 2:19-cv-00039-JRG-RSP
JUNIPER NETWORKS, INC. Defendants.	§ § §	Case No. 2:19-cv-00037-JRG-RSP

PLAINTIFF IMPLICIT, LLC'S
OPENING CLAIM CONSTRUCTION BRIEF

TABLE OF CONTENTS

TABLE OF AUTHORITIES

Cases

Pursuant to Local Patent Rule 4-5(a), Plaintiff Implicit, LLC respectfully submits this Opening Claim Construction Brief.

I. INTRODUCTION

The Implicit Patents disclose and claim a foundational technology to demultiplex (*i.e.*, separate out) many different types of network traffic and inspect that traffic to determine the type of application protocol (*e.g.*, HTTP) and application data (*e.g.*, Facebook, Skype, DDoS attacks, viruses) that the traffic is carrying. The constructions that Implicit proposes stay true to Implicit's invention. The Defendants, in contrast, seek to add limitations to the claims that are not supported by the specification. These proposed constructions result in the claims covering virtually no real-world network computer system, including the systems that Implicit actually designed and built on the patented technology. The purpose of those constructions is not to assist the fact finder in resolving the infringement disputes, but to allow the Defendants to assert noninfringement on grounds unrelated to Implicit's inventions. Implicit respectfully requests that the Court reject the Defendants' constructions and enter Implicit's proposed constructions.

Defendants also contend that collateral estoppel from the *NetScout* claim construction applies. That is incorrect. A number of terms below—such as "one or more routines"—were not at-issue in *NetScout* at all. Even for the same claim terms, the issues and the evidentiary records are different here than in *NetScout*. The Court also construed multiple terms in *NetScout*—not just one term. The stipulation in *Sandvine*, for example, lists approximately 12 claim terms; those terms did not all present one issue. The verdict in *NetScout* was a single-question infringement verdict, and NetScout raised multiple noninfringement arguments at trial. In these circumstances, none of the claim constructions from *NetScout* were critical or necessary to the *NetScout* and *Sandvine* judgment under Fifth Circuit collateral estoppel law. *Blitzsafe Tex., LLC v. Honda Motor Co.*, 2016 U.S. Dist. LEXIS 123572, at *24 (E.D. Tex. Sep. 12, 2016); *see also Novartis Pharms.*

Corp. v. Abbott Labs., 375 F. 3d 1328, 1334 (Fed. Cir. 2004) (finding noninfringement verdict was not preclusive where "[t]he verdict form on which the jury indicated its decision did not require the jury to specify the one or more limitations in claim 1 that it found [Defendant's] product did not contain" and "there is no record evidence explaining the jury's rationale for its verdict"). The *NetScout* claim construction is currently on appeal in the *Sandvine* case and is in the post-trial motions stage in the *NetScout* case, and the claim constructions will be reviewed *de novo*. In these circumstances, Implicit respectfully requests that the Court decline to apply collateral estoppel and construe these terms in the manner proposed by Implicit.

II. BACKGROUND AND TECHNOLOGY OVERVIEW

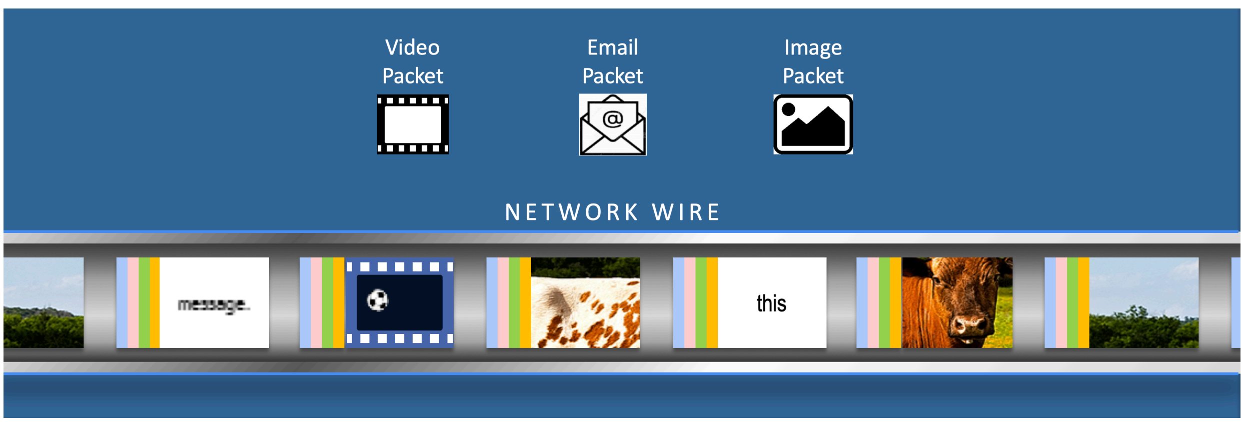
This case involves six related patents that are referred to collectively as the Demultiplexing Patents (or "the Implicit Patents").[1] The Patents are entitled "Method and System for Data Demultiplexing," and provide an architecture for demultiplexing data into different messages (or flows) to inspect and analyze each flow at the application layer. Traffic on a network (such as the Internet) is multiplexed together, which means that the traffic from different messages (such as video, image, or email) are combined and transmitted over a medium. Demultiplexing refers to the process of separating out that combined traffic back to identify and inspect packets from each message.

Network traffic is transmitted in smaller units called "packets" that contain smaller pieces of the overall message. Computers communicate using a layered "stack" of protocols, and each packet has a number of headers that generally correspond to each protocol in the stack, *e.g.*, TCP/IP is a common protocol stack for Internet communications that uses the TCP and IP protocols. There

[1] The following are the Demultiplexing Patents at issue in this case: U.S. Patent Nos. 8,694,683 ("the '683 Patent"); 9,270,790 ("the '790 Patent"); 9,591,104 ("the '104 Patent"); 10,027,780 ("the '780 Patent"); 10,033,839; ("the '839 Patent") and 10,225,378 ("the '378 Patent").

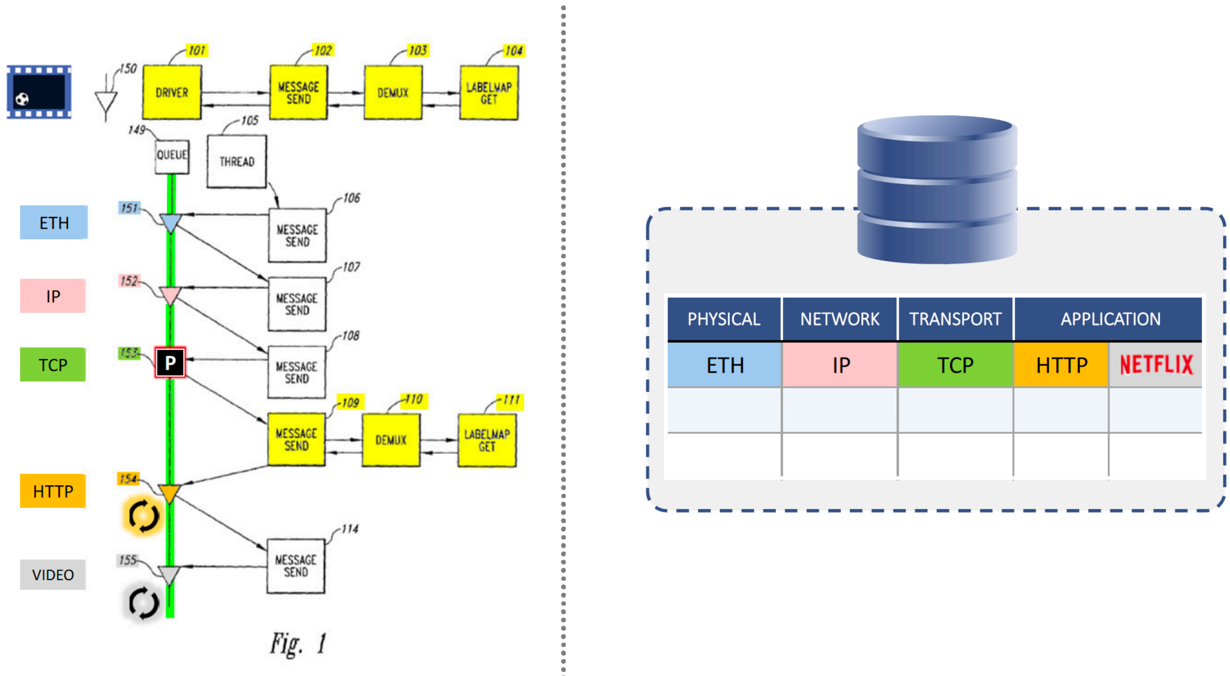
are protocols relating to the data link that the traffic travels on (*e.g.*, Ethernet). There are also protocols for applications that the data is associated with. Common examples are HTTP or HTTPS for web-based traffic and SMTP for email. The data itself (*e.g.*, the video, image, or email) is on top of the protocol stack in the packet. As the packets travel along a network like the Internet, packets from multiple messages are mixed together on the network, resulting in a multiplexed stream of packets. *See generally*, Ex. 1, at ¶¶ 35–57, 99–108 (describing network processing and how network software process packets up and down the protocol stack).

A multiplexed stream of packets for three different messages—video, email, and an image, is shown in the figure below:



The Demultiplexing Patents provide a path-based architecture that inspects the packets at the application layer to demultiplex a received multiplexed stream. As a packet arrives, the software inspects information in that packet in the lower-level protocols. Packets in a message are typically associated together based on the addresses of the source and destination of the message (typically the IP addresses), the network port from which the message originated, and the network port to which the message is destined. If the packet is for a message that has not yet been identified, the software creates the path for that message.

The software routines then inspects the packet at the higher-level protocols and continue to create the "path" as the software inspects and identifies the application-level protocol for the message (*e.g.*, HTTP) and the application data (*e.g.*, Netflix). This example process is shown in the annotated version of Figure 1 of the Demultiplexing Patents for Netflix traffic.

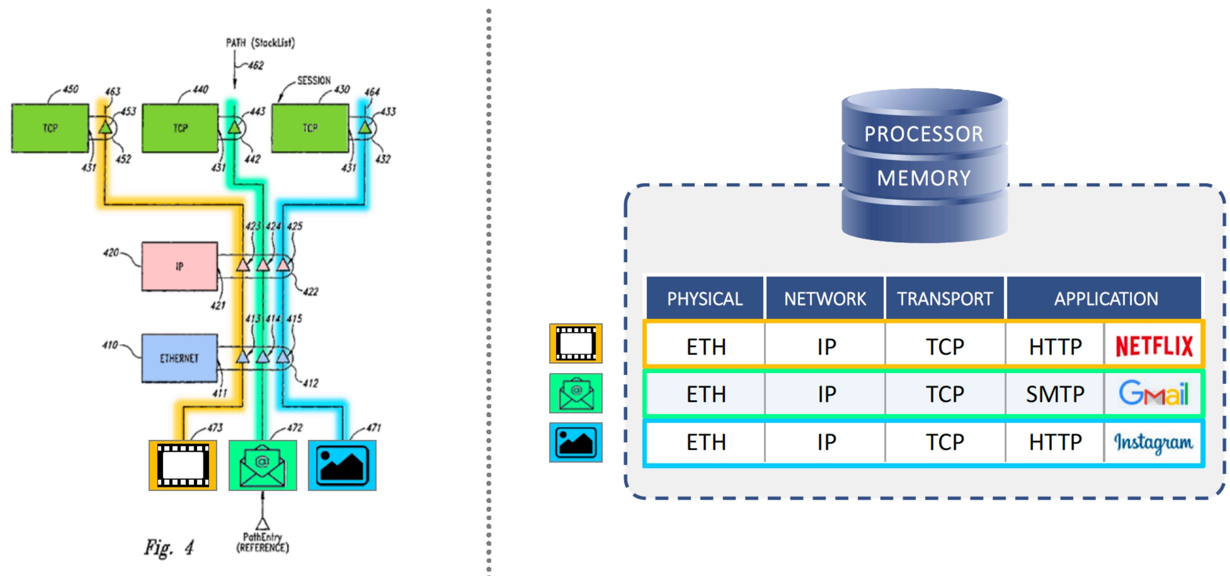


While this example shows the software identifying the application protocol (HTTP) and the application data (Netflix) based on a single packet, it often takes more than one packet for the system to identify the application protocol and the application data. This is because that application information may span multiple packets. It thus may take more than one packet to definitively identify the application protocol and application. In those cases, the software will continue to create the path based on inspecting that group of packets (including ensuring that they are in the right order).

The process to create the path requires more complicated programming and analysis at the application protocol and application stages. The Implicit Patents overcome this speed problem by providing a way to make the complicated branching decisions for the initial packet or packets of

the flow, store that path, and then use that path when the software processes subsequent packets of the flow.

After the software identifies the path for a message, it stores that path for that message (typically based on the address and port information that uniquely identifies that message). Then, when subsequent packets from that message arrive, the software processes those packets using the path that it has already built to handle that message. This processing is fast because it does not need to re-create the path. The example below from Figure 4 of the Patents shows the stored path data structure (typically a flow table) for three messages—a Netflix video, an email for Gmail, and an image for Instagram:



Claim 1 of the ’683 Patent shows these three concepts: create the path, store the path, and process subsequent packets using that path, where the path indicates that the software can inspect at the application layer (the “execute . . . to convert one or more packets having a TCP format into a different format” limitation):

> A first apparatus for receiving data from a second apparatus, the first apparatus comprising:

a processing unit; and a

memory storing instructions executable by the processing unit to:

> create, based on an identification of information in a received packet of a message, a path that includes one or more data structures that indicate a sequence of routines for processing packets in the message;
>
> store the created path; and
>
> process subsequent packets in the message using the sequence of routines indicated in the stored path,
>
> wherein the sequence includes a routine that is used to execute a Transmission Control Protocol (TCP) to convert one or more packets having a TCP format into a different format.

The invention results in a dynamic system to create a path when the packet arrives and then store those paths for future use.

III. AGREED CONSTRUCTIONS

The Parties have reached agreement on the following constructions and request that the Court enter an Order adopting same.

Term	**Agreed Construction**
"message"	"a collection of data that is related in some way, such as stream of video or audio data or an email message."
"state information"	"information that is specific to a software routine for a specific message, that can be used for all packets of the message, and that is not information related to an overall path"
"the packet of the message"	"the one or more received message packets used to create a path"
"key [value]"	"information that can be used to identify the session of a protocol," and—as used in the '104, '780, '839, and '378 Patents—the determine/determining operation/step is performed before the identify/identifying operation/step

IV. ARGUMENT

Implicit requests that the Court enter its proposed constructions in this case. The Defendants' proposed constructions are laced with noninfringement defenses that are divorced from the inventions described in the Implicit Patents and would have the Patents cover virtually no real-world network computer system—including the types of systems that Implicit actually designed and built based on its patented technology. Implicit's proposed constructions are grounded in the plain-and-ordinary meaning of these terms as those in the computer and networking fields would have understood them, comport with the intrinsic record, and focus and clarify the claim language to allow a jury to focus on the merits of whether Defendants (or the prior art) actually practice the invention as disclosed and claimed.

A. "Sequence of [Two or More] Routines"

Plaintiff's Proposed Construction	Defendants' Proposed Construction
"an ordered arrangement of [two or more] software routines that was not identified (i.e., configured) prior to receiving a first packet of a message"	"[two or more] software routines arranged in a sequence that was not established in a chain of modules connected before receiving a first packet of the message"

This term is related to the claimed "path" in three of the Implicit Patents (the '683, '790, and '104 Patents) and how the software "creates" that "path." Implicit's proposal captures the basic concept in the claims that the "path," which is a data structure, is created after the first packet arrives, *i.e.*, the path was not preconfigured before the first packet arrived. This proposal is virtually identical in substance to the construction that this Court and Judge Illston in the Northern District of California arrived at when addressing this issue in a number of prior Implicit cases (*F5 Networks I*, *F5 Networks II*, and *Trend Micro*).[2] The parties even agreed to that construction in

[2] The following are each of the Claim Construction Opinions that have addressed the "sequence of [two or more] routine" terms and similar terms. *Implicit, LLC v. NetScout, Inc.*, No. 2:18-CV-

the *Palo Alto Networks* case,[3] and the Court adopted the agreement. The Court should adopt that same construction here.

The Defendants, in contrast, propose a construction that has never been adopted in any prior case. The Defendants' construction adds a new limitation to the claims—that the sequence of routines that form the path must "not [be] established in a chain of modules connected before receiving a first packet of the message." The Court should reject this limitation.

The issue that the parties raise here relates to the nature of the "path" in these claims: what is a "path" and when does the software "create . . . a path?" The claims themselves answer these questions. The "path" includes a data structure, and that data structure indicates the "sequence of routines for processing packets in a message," such as a video, email, or image. *E.g.*, '683 Patent, claim 1. The claims also speak to when the software creates the path: ***after*** it receives the first packet of a message. This logically flows from the language in claim 1 of the '683 Patent: the software creates the path "based on an identification of information in a received packet of a message." ***Before*** receiving the first packet in the message, there is no completed path in a data structure for that message; it is only ***after*** the first packet in the message that the software creates the path. That data structure indicates the sequence of routines for processing subsequent packets in the message.

00053, Dkt. 111 (E.D. Tex. Apr. 15, 2019) (Payne, M.J.) ("*NetScout*"); *Implicit, LLC v. Huawei Techs. USA, Inc.*, No. 6:17-CV-182, Dkt. No. 101 (E.D. Tex. Mar. 6, 2018) (Gilstrap, J.) ("*Palo Alto Networks*"); *Implicit, LLC v. Trend Micro, Inc.*, No. 6:16-CV-80, Dkt. No. 115, 2017 WL 1190373 (E.D. Tex. Mar. 29, 2017) (Gilstrap, J.) ("*Trend Micro*"); *Implicit Networks, Inc. v. F5 Networks, Inc.*, No. 3:14-CV-2856, Dkt. No. 57, 2015 U.S. Dist. LEXIS 60197 (N.D. Cal. May 6, 2015) (Illston, J.) ("*F5 Networks II*"); *Implicit Networks, Inc. v. F5 Networks, Inc.*, No. 3:10- CV-3365, Dkt. No. 93, 2012 U.S. Dist. LEXIS 27238 (N.D. Cal. Feb. 29, 2012) (Illston, J.) ("*F5 Networks I*").

[3] Counsel for Fortinet was counsel for Palo Alto Networks in that case.

The specification of the Implicit Patents supports the plain language. The Patents teach that existing network computer systems "typically use[d] predefined configuration information to load the correct combination of conversion routines for processing data." '683 Patent, at 1:48–59. In these systems, the path for each type of message was pre-created ***before*** the packets arrived; they did not create any part of the path data structure ***after*** receiving the first packet of a message.

The main problem with these prior approaches is that they consumed resources or were too static in their processing to be useful for anything except specific-purpose-built devices (*e.g.*, remote controls or cameras). The potential combinations required to process each possible type of message a system may encounter (*e.g.*, each type of email, video, or other flow of related packets) increases exponentially as the number of message types increases due to the stacked nature of network protocols and applications (*e.g.*, Facebook runs on top of HTTP, which runs on top of TCP, which runs on top of IP, which runs on top of Ethernet). Similarly, while a system could attempt to inspect every packet and determine for each packet which branches in the source code to take, this model would become computationally expensive as well. This is because the number of branches would grow with the number of applications. The end-result, as recognized in the Patents, is that "[t]he overhead of statically providing each possible series of conversion routines is very high." '683 Patent, at 1:48–59. This was a significant problem because state-of-the-art computers in the 1990s had less resources available than a modern-day smartphone.

The solution in the Implicit Patents was to create the path after receiving the first packet of a message—and to do so based on information in the packet. While common sense dictates that the system includes the software routines and the ways they could be connected together beforehand (that is how all computers work), the software does not create the actual path in the data structure until after it receives and processes the first packet of a message. This allows the

software to discover early in the demultiplexing process which routines are needed to process the data and then store the identification of these routines in a data structure so subsequent packets can benefit from these branching decisions without having to make them again. The path is the manifestation of the branching decisions through the set of all possible routines.

The prosecution history supports Implicit's construction. Implicit discussed during prosecution a prior art 1997 Ph.D. thesis by David Mosberger that described an operating system called SCOUT. The SCOUT operating system concept was intended for special-purpose devices that handled a limited amount of traffic types, such as television set-top boxes, remote controls, and cameras. Ex. 2, at 16–18. Because of this limited application, a SCOUT programmer could pre-configure the software for a particular device to handle the expected traffic (*e.g.*, configure a path for MPEG video delivered to a set-top box or configure a path to handle JPEG images sent from a camera). *See id.* In other words, SCOUT loaded a configuration that codified all the branching decisions that the software would need to make. SCOUT then used that configuration to create all the possible path data structures ***before*** the packets associated with those paths were received. In this way, SCOUT allowed the branching decisions to be avoided, but it did so by statically defining and limiting the number of those decisions that could be made by the system. The Implicit system, in contrast, is intended for general network devices (such as switches, gateways, and firewalls) whose purpose is to handle many different types of Internet traffic and applications and to be able to demultiplex all of that traffic.

Figure 3.6 from Dr. Mosberger's thesis shows an example of the limited number of pre-configured paths that SCOUT utilized. The figure shows with green lines (p1, p2, p3, p4, and p5)

the pre-created "paths" in one implementation, namely a device that handles traffic based on certain types of lower-level protocols (*e.g.*, Ethernet, ARP, IP, UDP, and TCP), Ex. 2, at 87:

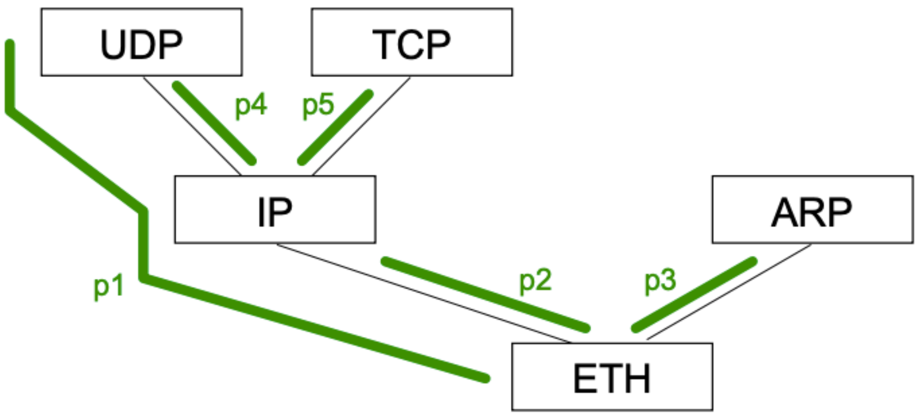


Figure 3.6: Paths Versus Classifiers

In this diagram, the green lines reflect the "paths," not the gray lines. Those green "paths" are created by the system prior to SCOUT receiving the first packet of a message. *See NetScout*, at 11–13 (reproducing prosecution history). Then, as the packets arrive, SCOUT identifies which of those pre-stored green paths (p1, p2, p3, p4, or p5) to use to handle the traffic. *See id.*; *see also* Ex. 2, at 38 (showing example path to display MPEG video). SCOUT does not create any paths after receiving the first packet of a message.

Based on these straightforward disclosures, Implicit believes that the Court could decline to construe these terms because the plain language of the claims unambiguously describes the "path," and how it operates is sufficiently clear. However, given the long history with patents within the family of the Implicit Patents, Implicit has proposed its construction above to match how prior courts have construed the term (and how Implicit proposed that term to be construed in the *Palo Alto Networks* and *NetScout* cases): that the ordered arrangement of routines in the path data structure is not "identified (i.e., configured) prior to receiving a first packet of a message."

The Defendants' construction introduces a new limitation into the claims—"a chain of modules connected before receiving a first packet of the message." The Implicit Patents and their claims do not recite or disclose a "chain of modules" or when they are "connected." And it is uncertain what Defendants' construction actually means. To the extent it is just another way of saying that the path cannot be created until after the first packet of a message arrives, it is redundant of the existing claim language and unhelpful to a jury. And to the extent it shifts the inquiry from (1) whether the path ***was created*** for a particular message before the first packet of the message arrived to (2) whether the path ***could have been created*** using source code modules that were written before the system begins receiving packets, the construction is incorrect because it conflicts with the plain language of the claim as a whole, which requires that the path be created "***based on*** an identification of information in a ***received packet*** of a message." *See, e.g.* '683 Patent, claim 1.

Neither Implicit's prosecution history concerning Dr. Mosberger's thesis nor the thesis itself disclaims the existence of software routines prior to receiving the first packet of a message, nor could it. Nor do they disclaim systems in which the source code indicates how the branching decisions in the source code might indicate the sequence of processing steps to process packets within a message (which would have the absurd result of excluding virtually every networked computer as a practical matter). Indeed, reviewing the prosecution history related to SCOUT, courts have reached two key conclusions that weigh against adopting such a construction:

- "The patentee did not disclaim the *existence* of software routines prior to receiving a first packet of the message." *NetScout*, at 13 (emphasis in original).
- "Implicit did not disclaim the ability to create a sequence of conversion routines by relying in some part on predefined 'configuration information,' but only the use of pre-configured paths." *F5 Networks I*, at 6.

Such a construction would eviscerate the essence of Implicit's patented invention. Indeed, that type of shift conflicts with the intrinsic record above and it is significant. It would allow

Defendants to argue that that they do not infringe on the basis that their source code establishes all of the modules and how they could be connected. That position would exclude the embodiments of the Implicit Patents, a highly disfavored result. *Medrad, Inc. v. MRI Devices Corp.*, 401 F.3d 1313, 1320 (Fed. Cir. 2005) ("A claim construction that does not encompass a disclosed embodiment is... rarely, if ever, correct.") (alteration in original) (internal quotation marks and citation omitted).

Implicit respectfully requests that the Court reject this source-code-based claim-construction noninfringement defense. The source code is not the "path" of the claims—the "path" is the actual data structure created and stored to indicate the sequence of routines to process subsequent packets of a message. While the source code might include many branching decisions, these decisions are not known until the software inspects a given packet. Only after this inspection does the software described in the Implicit Patents create a data structure capturing these branching decisions. Using that path data structure, subsequent packets of the message will be processed without having to make these branching decisions again. The source code is not the "path"; the data structure storing the results of the processing decisions through the source code is the claimed "path."

Nevertheless, Implicit expects the Defendants to raise this defense because NetScout recently prevailed at trial and raised this improper argument. NetScout asserted, among other things, that it did not infringe because all of its "paths" were already programmed into the if-then-else statements in its source code. The source code, however, is not the claimed path. The path is the data structure(s) that indicates the sequence of routines, and which is created based on information in a packet received by the device. Whether that path is pre-created is the merits question—not whether the writing of the source code pre-dates the arrival of the first packet.

For these reasons, Implicit respectfully requests that the Court adopt Implicit's proposed construction, reject Defendants' proposed construction, and reject Defendants' claim-construction defense that the "path" is the source code.

B. "List of Conversion Routines"

Plaintiff's Proposed Construction	**Defendants' Proposed Construction**
"an ordered arrangement of [two or more] software routines that was not identified (i.e., configured) prior to receiving a first packet of a message"	"[two or more] software routines arranged in a sequence that was not established in a chain of modules connected before receiving a first packet of the message"

The Parties' competing proposals for "list of conversion routines" present the same issue for resolution as the prior term. Resolution of the prior term should resolve this dispute as well, and Implicit incorporates those arguments by reference.

C. "One or More Routines"

Plaintiff's Proposed Construction	**Defendants' Proposed Construction**
Plain and ordinary meaning. No construction necessary.	"[two or more] software routines arranged in a sequence that was not established in a chain of modules connected before receiving a first packet of the message"

The term "one or more routines" should be given its plain-and-ordinary meaning. This term has not been construed before in any of the prior cases. This term was not at issue in those prior cases. These terms appear only in the'780 Patent, '378 Patent, and '839 Patent, three of the six Patents asserted here. Those patents, which recently issued, were not asserted in those prior cases.

Implicit therefore requests that the Court decline to further construe the "one or more routines" term. There is nothing about the term that requires further explanation for a jury (other

than, perhaps, that these are software routines). The surrounding claim language makes clear that the one or more routines are identified and then, the path is created using those routines that have been identified. *See* '378 Patent, claim 1 ("identifying . . . one or more routines for processing the packet . . . creating a path using the identified one or more routines"); '780 Patent, claim 1 ("identifying . . . one or more routines for processing the packet . . . creating a path using the identified one or more routines"); '839 Patent, claim 1 ("identifying . . . one or more routines for processing the packet . . . creating a path using the identified one or more routines"). No further construction is necessary to assist the jury.

The prosecution history of these patents also supports Implicit's position. During prosecution of these patents, Implicit expressly disavowed the statements that formed much of the basis of the construction of the "sequence of [two or more] routines"-related terms *F5 Networks I* and *F5 Networks II*. Ex. 3, at 7–10 ('780); Ex 4, at 4 ('839); Ex. 5, at 7–10 ('378). During prosecution of the '780 Patent, for example, Implicit explained that "the actual routines themselves," as well as "information that lists and orders these routines, may exist prior to 'receiving' the 'packet'" under the claims:

> Applicant wishes to clarify that the "identifying" of "one or more routines for processing the packet" in claim 26 [which issued as claim 1] can, in some cases, rely in part or entirely on pre-identified information (*e.g.*, a sequence of routines). ***Thus, the "identifying" of claim 26 might include reading a pre-defined sequence of routines from a file. On the other hand, claim 26 makes clear that the "path" is created after "receiving ... a packet of a message" since it is created "using the identified one or more routines.***
>
> Thus, as just one example, claim 26 would cover a method in which, after receiving a packet of a message, one or more routines (*e.g.*, a TCP routine and an HTTP routine) are identified by using the "key value" to look up pre-defined information, such as an ordered list of routines found in a configuration file or lookup table. The identified routines may then be used to create a "path" for the packet. ***Thus, the actual routines themselves (the TCP routine and the HTTP***

> ***routine, in this example) as well as information that lists and orders these routines, may exist prior to "receiving" the "packet" of claim 26. For the "path," on the other hand, the language of claim 26 requires that this is necessarily created after receiving the "packet,"*** at least because the path is created using "one or more routines" that are identified "using the key value" that is "determined based on one or more headers in the packet."

Ex. 3, at 9 (emphases added). These statements also confirm that the "path" is something altogether different than the actual source code routines or how they could be programmed to be ordered together (*e.g.*, the source code)—and that the claims require that the path be created after receiving the first packet of the message. *Id.*

The Defendants propose the same construction for this term as they do for the "sequence of [two or more] routines" terms. As a result, the Defendants' construction here suffers from the same problems discussed above. And, to the extent the Defendants' construction excludes from the scope of the claims what the prosecution history for these patents expressly state is covered, it should be rejected for that reason as well.

That approach would, like above, erase the distinction between two key concepts—(1) the "routines" and how they may be ordered, and (2) the actual "path" that is created in a data structure. Erasing that distinction will allow for an argument at trial that the "path" is the if-then-else trace in the source code—and not the actual data structure(s) created after the first packet arrives. All computers use if-then-else-case statements and logic. As detailed above, Implicit requests that the Court expressly reject this source-code-based argument at this stage. Implicit also requests that the Court decline to further construe this term.

D. The "Execute a Transmission Control Protocol (TCP) to Convert One or More Packets Having a TCP Format Into a Different Format" Terms

Plaintiff's Proposed Construction	Defendants' Proposed Constructions
Plain and ordinary meaning. No construction necessary. Alternatively, "a software routine for processing the packet from the TCP layer to another layer in the protocol stack"	**Execute**: "operate on one or more packets whose outermost header is a TCP header" **Convert**: "convert the outermost header structure of the packet(s) from TCP to another type of header structure"

The disputes for each of these terms[4] raise three issues: (1) whether the concept of an "outermost header" and an "outermost header structure" should be imported into the claim language; (2) whether a pointer can define the outermost header of a packet; and (3) whether advancing a reference can meet the claimed "conversion." Implicit addresses each of these issues below.

1. The Court Should Decline to Import the "Outermost Header" Concept Into All Claims of the Implicit Patents

Implicit proposes that these terms should be afforded their plain-and-ordinary meaning, which refers to processing packets up and down the protocol stack, typically from the TCP layer

[4] The "execute" and "convert" terms include the following: "a routine that is used to execute a Transmission Control Protocol (TCP) to convert one or more packets having a TCP format into a different format"; "a routine that is executable to perform a Transmission Control Protocol (TCP) to convert at least one of the packets of the message into a different format"; "a routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format"; "a particular routine that is used to execute a Transmission Control Protocol (TCP) to convert packets having a TCP format into a different format"; "a session associated with a transport layer protocol that is executed to convert one or more packets in a transport layer format into a different format"; "a second routine that is used to execute a second, different protocol to convert packets of the different format into another format"; "a [third] routine that is used to execute a [third], different [application layer] protocol to further convert the packets"; "a routine that is used to execute a Transmission Control Protocol (TCP) to process packets having a TCP format"; "a routine that is used to execute TCP to process at least one of the subsequent packets having a TCP format"; and "a second routine that is used to execute a second protocol to process packets having a format other than the TCP format, wherein the second protocol is an application-level protocol."

(or the transport layer) to another layer in the protocol stack, such as the application layer. The ordinary meaning of a "format" of a packet in connection with network processing is which layer of the packet is being processed. Ex. 1, at ¶¶ 109–56.[5] Thus, "converting" packets from one format to another format would be understood by those skilled in the art as processing the packet from one layer in the protocol stack to another layer in in the protocol stack, *e.g.*, from the TCP layer to the application layer. *Id.* at ¶¶ 155–56; *see also id.* at ¶¶ 99–107, 114–44 (describing how systems convert packets "down" the protocol stack for transmission and "up" the protocol stack upon receipt). These concepts flow from the claim language, which contains understandable terms such as "execute," "convert," and "format." The language does not restrict how that conversion processing occurs—whether through the use of pointers, using copies of the packets, other data structures, or other ways.

Rather than afford those terms their plain meaning, the Defendants' constructions replace those terms with the more complicated concepts of an "outermost header" and "outermost header structure" of a packet. The claims themselves show that adding these concepts to all of the claims is, at most, unnecessary. "Differences among claims can also be a useful guide in understanding the meaning of particular claim terms," and the presence of a dependent claim that adds a particular limitation "gives rise to a presumption that the limitation in question is not present in the independent claim." *Phillips v. AWH Corp.*, 415 F.3d 1303, 1314–15 (Fed. Cir. 2005 (en banc).

[5] Juniper has moved to exclude the Almeroth Declaration in connection with its Motion to Disqualify Dr. Almeroth. Implicit opposes that Motion for the reasons that are (and will be) laid out in its briefing. Even if the Court were to grant Juniper's Motion, however, Implicit submits that the Court can consider the declaration in these consolidated proceedings that involve Fortinet and Imperva. Fortinet and Imperva have not objected to the Declaration. And, in any event, the Court can certainly consider the intrinsic and extrinsic evidence cited within Dr. Almeroth's declaration and discussed in and attached to this brief.

Here, some claims of the Implicit Patents already include the concept of an outermost header in the claims while other claims—including an independent claim—lack that limitation. *Compare, e.g.*, '683 Patent, claim 16 (reciting "a particular routine that is used to execute a protocol to ***convert packets from an input format to an output format***") *with* claim 20 (reciting the invention of claim 16 "wherein the particular routine is executable to ***convert packets by removing an outermost header of the packets***") (emphases added); *see also* '683 Patent, claim 24 ("***removing an outermost header*** of a given packet using a first routine corresponding to a protocol in a first layer and by ***removing the resulting outermost header*** using a second routine corresponding to a different protocol in a different layer") (emphases added). In fact, almost none of the claims of the Implicit Patents include the term "outermost header."

This difference between the claims strongly suggests that it is improper to import the "outermost header" concept into claims that expressly omit that concept. *Arlington Indus., Inc. v. Bridgeport Fittings, Inc.*, 632 F.3d 1246, 1254–55 (Fed. Cir. 2011) (declining to import a "split" modifier for a metal adaptor where some claims expressly recited "split"-related limitations and other claims omitted those limitations on the adapter). In addition, the "outermost header" language does not appear in the specification of the Implicit Patents. And none of the claims in 5 out of the 6 Implicit Patents asserted here recite "outermost header." This further weighs against importing this concept into all of the claims—especially when the patentee knew how to direct claims to the outermost header of packets and did so in certain claims of the '683 Patent. For these reasons, Implicit respectfully requests that the Court decline to import an "outermost header" limitation into all of the claims, and instead, construe the term according to its ordinary meaning.

In addition, as outlined below, Implicit respectfully requests that the Court reject Defendants' positions that a pointer cannot determine the "outermost header" of a packet and that

using pointers to process packets up the protocol stack is not conversion from one format to another. The evidence shows that these restrictions on the claims are incorrect.

2. A Pointer Can Determine the Outermost Header of a Packet

A pointer to a header can determine the "format" of a packet, including by identifying the "outermost header" of the packet. At the time the Implicit Patents were filed, using pointers was the common way that operating systems determined the format of the packet to process it up the protocol stack. One of ordinary skill in the art would thus have understood that the claims of the Implicit Patents encompassed using pointers to determine the format of a packet at each layer in the protocol stack, reflected in the evidence addressed above. *See* Ex. 1, at ¶¶ 150-156. Nothing in the intrinsic record excludes from the scope of the Implicit Patents using a pointer to determine the format of packet.

The evidence confirms that the claims of the Implicit Patents do encompass using a pointer to a protocol header to determine the "format" of the packet (including the "outermost header"). The references cited on the face of the Patents confirm this basic truth.[6] By 1998, "[t]raditional protocol implementations usually overla[id] a pointer to a header structure on top of a buffer and then access[ed] that header's fields through typecasting and pointer de-referencing." Ex. 6, at 3; *see also* Ex. 1, at ¶¶ 110–13. These implementations were usually written in the C-based programming languages, one of the most common programming language families at the time (as well as today).[7] *See* Ex. 1, at ¶¶ 58–61. These systems make heavy use of pointers to simplify

[6] The references cited on the face of the Implicit Patents are intrinsic evidence. *Formation, Inc. v. Benetton Grp. SpA*, 401 F.3d 1307, 1311 (Fed. Cir. 2005) ("This court has established that 'prior art cited in a patent or cited in the prosecution history of the patent constitutes intrinsic evidence.'").
[7] Much of Implicit's software implementing the inventions was written in C-based languages, as is the packet-processing software in the Defendants' products.

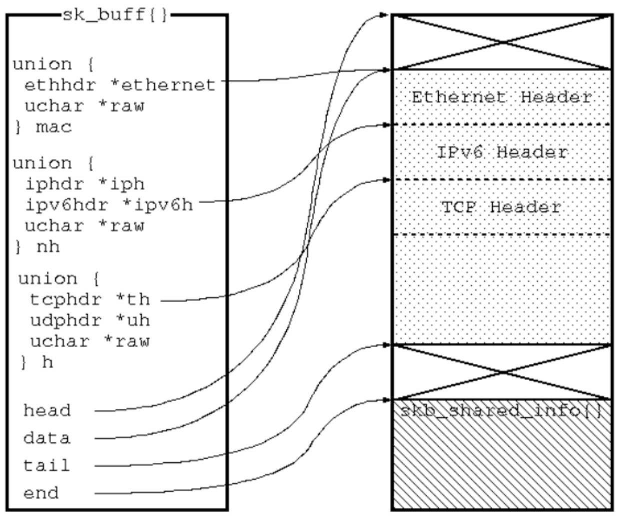
header processing at each layer, *e.g.*, by using pointers, "[r]emoving data from a packet is a simple, straightforward operation involving pointer manipulation rather than data copying":

> ***Protocols implemented in traditional system programming languages like C make heavy use of memory pointers***. Pointers are used in a variety of ways in order to avoid costly memory copying. Pointers reduce data copying because abstract data types (e.g., buffers, protocol headers, and state information) can be passed by reference instead of value; pointers simplify header processing because structures defining headers are usually typecast onto data buffers, allowing the individual fields to be accessed randomly instead of sequentially. ***For example, protocol data (a packet) is normally passed from protocol to protocol in the form of a memory buffer referred to by a pointer variable. Removing data from a packet is a simple, straightforward operation involving pointer manipulation rather than data copying.*** Likewise, when data is added to a packet, memory buffers are normally chained together using pointer manipulation instead of data copying.

Ex. 7, at 3 (emphases added); *see also* Ex 8, at 21:15–30 (describing software that would "get a pointer to start of protocol header, call layer parse routine, determine the protocol at next level, [and] set pointer to start of next layer protocol"); Ex. 1, at ¶¶ 111–13.

These example implementations of protocol processing included systems based on Linux or FreeBSD, two very common open-source codebases. Those systems use pointers to convert up and down the protocol stack, similar to the manner described above. In Linux, for example, the operating system uses a buffer structure (called `sk_buff`) that creates pointers to each header in

the packet, such as an Ethernet header, IPv6 header, and TCP header shown in the diagram below (with the `ethhdr`, `ipv6hdr`, and `tcphdr` type pointer structures):[8]



As the processing converts packets up the stack, Linux-based systems can use the `skb_pull` function to advance the `head` pointer to the next location in the packet (*e.g.*, down from Ethernet to IP in the above figure) and then create pointers to the right type header structure (*e.g.*, creating an `ethhdr` pointer structure during Ethernet processing and then an `ipv6hdr` during IP processing).[9] FreeBSD similarly uses pointers to headers in a buffer (called `mbuf`) to access those headers and convert packets up the protocol stack. *See*, Ex. 1, at ¶¶ 113–124 (discussing Linux and FreeBSD and how they operate).

The prosecution history evidence also supports rejecting Defendants' interpretation of "outermost header" as excluding the use of pointers. *See, e.g.*, *NetScout*, at 26. In a reexamination of a patent related to the Implicit Patents, U.S. Patent No. 7,711,8507, Dr. Ng (Implicit's expert)

[8] The image below is available from http://www.skbuff.net/images/skbuff.png.

[9] The basic Linux functions for the `sk_buff` structure are explained at http://www.skbuff.net/skbbasic.html.

submitted a declaration related to a reference known as Decasper. *See* Ex. 9, ¶ 4. Decasper disclosed software functionality for a router. *Id.*

In Dr. Ng's declaration describing Decasper, he explained that "the outermost header is always IP" in Decasper. *Id.* at ¶ 6; *see also* Ex. 1, at ¶¶ 134–35. The Decasper source code shows that the "outermost header" that Dr. Ng referred to was defined by a pointer to the IP header. The code creates a pointer (`*ip`) to the IP header structure, exemplified by the declaration below for IPv6 traffic:

```
struct ipv6 *ip;
```

Ex. 10, at DEFSPA182329; *see also* Ex. 1, at ¶¶ 125–35 (discussing Decasper source code); *id.* at ¶¶ 69–73 (discussing pointers in C); Ex. 11, at 93 (discussing pointers in IBM C/C++ Language Reference). This source code shows that the "outermost header," here the IP header as identified by Dr. Ng, may be determined by a pointer to a header within the packet.

Decasper was not unique in its use of pointers among the references that are part of the intrinsic evidence. SCOUT (Mosberger) also used pointers to determine the outermost header of the packets as the system converted the packets up the protocol stack. Similar to Linux, SCOUT utilized a buffer with pointers to the two ends—identified as the "head" and "tail"—of the message portion of the buffer, as shown below (Ex. 12, at DEFSPA003055):

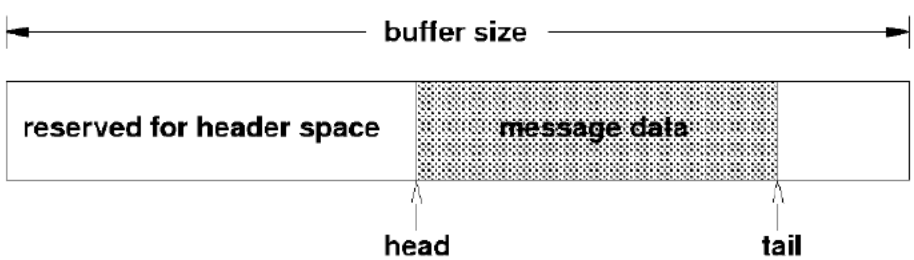


Figure 1: High-level view of a message.

To convert down the protocol stack to transmit a message (*e.g.*, convert from IP to Ethernet), SCOUT used the `MsgPush` function. That function decrements the `head` index pointer (moves it to the left in the above figure) and then copies the protocol header into that space. Ex. 12, at DEFSPA003055–56. To convert up the protocol stack upon receiving a message (*e.g.*, from Ethernet to IP), SCOUT used the `MsgPop` function. That function increments the `head` index pointer (moves it to the right in the above figure) to the next header in the message and then processes that header—even though the other lower-level headers would still remain in the buffer. *Id*. Those skilled in the software art recognize that this technique is commonplace in software written in the C programming language. *See, e.g.*, Ex. 1, at ¶¶ 58–91, 109–56.

The SCOUT source code confirms the presence of this functionality. For Ethernet processing up the protocol stack, for example, the source code increments the `head` index pointer by the length of the Ethernet header. Ex. 1, at ¶¶ 136-144 (discussing SCOUT source code). This converts the packet from Ethernet to IP for processing at the IP layer. *See id.*

All of this evidence shows that a pointer may—and oftentimes does—determine the format of a packet. To the extent that the Court retains the "outermost header" construction in connection with these terms, Implicit respectfully requests that the Court clarify that a pointer can be used to determine the outermost header of a packet and reject Defendants' contrary position.

3. Using Pointers to Process Packets Up the Protocol Stack, Including Advancing a Reference, Can Be the Claimed Conversion

The above evidence also shows that the claimed "convert" limitations, which reflect converting a packet from one format (*e.g.*, TCP) to another format (*e.g.*, an application format), are met using pointers to travel up and down the protocol stack for a message. As is well known to a person of ordinary skill in the art as reflected in the prior art, in these traditional systems, a packet can be "passed from protocol to protocol in the form of a memory buffer referred to by a

pointer variable." Ex. 7, at 3. Converting "down" the protocol stack, the protocol can chain together different memory buffers by manipulating pointers rather than copying data. *Id.* Converting "up" the protocol stack, each protocol can convert the packet by removing data using a "simple, straightforward operation involving pointer manipulation rather than data copying." *Id*. This is how, for example, Linux-based systems use the `sk_buff` structure detailed above. SCOUT, Decasper, and FreeBSD operate in a similar manner to convert "up" the protocol stack.

The Implicit Patents do not exclude this common, basic network-processing functionality. The Implicit Patents provide a path-based architecture that can inspect and demultiplex traffic at the application level—regardless of how the source code describes how the system converts from one layer to another (*e.g.*, from TCP to an application layer).

Should the Court require conversion of the "outermost header structure" to satisfy the "convert" limitations, Implicit respectfully requests that the Court clarify that a system can use pointers to a header to determine the outermost header of that packet pre- and post- conversion, *i.e.*, that manipulating pointers is one way to convert packets from one format to another. To hold otherwise would have the effect of excluding C-based systems from the scope of the claims. The evidence supports this clarification and resolution of that issue at this stage.

Implicit also respectfully requests that the Court find that advancing a reference can satisfy the "convert" limitations. The above evidence explains that "[p]ointers reduce data copying because abstract data types (e.g., buffers, protocol headers, and state information) can be passed by reference instead of value; pointers simplify header processing because structures defining headers are usually typecast onto data buffers . . ." Ex. 7, at 3. To convert up the protocol stack, SCOUT similarly increments `head` index pointer to the next header; the `sk_pull` function in Linux operates in a similar manner. Each of these systems convert from one format to another up

the protocol stack by advancing a reference. The Court's construction should not exclude this type of functionality, which skilled artisans would have understood as "conversion" by the time the Implicit Patents were filed.

Indeed, the Implicit Patents disclose a "single copy" embodiment in which a reference to the message can be passed to each conversion routine and those routines can "advance the reference past the header information for the protocol so that the reference is positioned at the next header." '683 Patent, at 14:10–16. The Court in *NetScout* rejected an interpretation of the "convert" limitations that would encompass "merely moving a reference" and held that this "single copy" disclosure was not an embodiment of "conversion" as recited in the claims because the Patents states that a conversion routine "may perform no conversion of the message." *NetScout*, at 26–27 (quoting '683 Patent, at 14:10–16).

Implicit respectfully submits that the Court's holding was incorrect. The conversion routines in the Implicit Patents are what perform the "conversion." While those routines may perform no conversion in certain circumstances (*e.g.*, logging), the conversion routines are the only routines disclosed in the Implicit Patents that actually perform conversion. And the Patents teach that a reference to the packet can be passed into those conversion routines, which then advance the reference to the next header. In any event, regardless of whether the disclosure is an embodiment of "conversion" on its own, the disclosure shows that skilled artisans would understand that the use of a reference pointer—including advancing a reference—certainly could be used in the conversion process.

The intrinsic evidence recited above, which was not highlighted in the *NetScout* claim construction proceedings, further supports this view that advancing a reference can constitute the claimed conversion. *See e.g.*, Ex. 7, at 3 ("Removing data from a packet is a simple,

straightforward operation involving pointer manipulation rather than data copying. Likewise, when data is added to a packet, memory buffers are normally chained together using pointer manipulation instead of data copying."); Ex. 12, at DEFSPA003055–56 (explaining that, in the normal case in SCOUT, "popping a header simply incrementing the head index and returning a pointer to the old location"); *see also* Ex. 1, at ¶¶ 114–18, 136–144 (describing operation of Linux and SCOUT buffer structures). Thus, Implicit also respectfully requests that the Court clarify that a system can advance a reference as part of converting a packet from one format to another format (*e.g.*, converting from a TCP format to an application format).

E. The "Session Associated With a [Transport Layer/Different] Protocol" Terms

Plaintiff's Proposed Construction	**Defendants' Proposed Construction**
Plain and ordinary meaning. No construction necessary.	"information that is specific to a software routine for executing a specific protocol, that can be used for all packets of the message, and that is not information related to an overall path"

These are also terms[10] that have not been construed prior to this case. These terms do not need additional construction. The term "session" is a well-known term that references a temporary instance of something like a "brainstorming session," "workout session," or "recording session." A "session" in computer networking generally means the same thing, a temporary instance of information exchange between two devices, like a "login session." The Implicit Patents generally use the term in that sense, and the term does not require further construction.

[10] The "session" terms include the following: "a session associated with a transport layer protocol that is executed to convert one or more packets in a transport layer format into a different format"; "session associated with a [transport layer/different] protocol"; "a TCP session associated with [the received] one or more [received] packets"; "a single TCP session"; and "sessions corresponding to [various/respective] ones of the sequence of [two or more] routines."

The Defendants' construction requires two steps, each of which is inappropriate under *Phillips*. It first takes the description of a "session" in one embodiment and then limits the claims to that embodiment—even though the Implicit Patents lack the type of clear definitional language that Federal Circuit law requires to limit the claims in that way. *Thorner v. Sony Computer Ent't Am., LLC*, 669 F.3d 1362, 1365–66 (Fed. Cir. 2012) (reciting the exacting standard to find lexicography or disavowal). In one embodiment, the Implicit Patents teach that the "session" includes "protocol and state information associated with that instance of the protocol." '683 Patent, at 5:43–46.

The Defendants' construction then goes further. It takes that description of a "session" from one embodiment as including protocol-associated "state information" and replaces the term "state information" with the agreed construction of a different "state information" term—"state information ***associated with the message***." '104 Patent, claim 1 (emphasis added). This is also improper. While state information associated with a protocol can certainly overlap with state information associated with a message (*e.g.*, state information concerning whether the connection using the protocol is open, closed, or in the process of transmitting the message), there is no reason to collapse the two terms into the same definition, especially in the absence of a clear statement in the record that equates the two things. *Bd. of Regents of the Univ. of Tex. Sys. v. BENQ Am. Corp.*, 533 F.3d 1362, 1371 (Fed. Cir. 2008) ("Different claim terms are presumed to have different meanings.") (citation omitted). The Court should reject the Defendants' double-twisted construction and afford the "session" terms their plain meaning.

V. CONCLUSION

For the foregoing reasons, Implicit respectfully requests that the Court enter an order adopting Implicit's proposed constructions and rejecting the positions advanced by Defendants in their constructions.

Dated: January 28, 2020

Respectfully submitted,

By: /s/William E. Davis, III
William E. Davis, III, *Lead Attorney*
Texas State Bar No. 24047416
bdavis@bdavisfirm.com
Christian J. Hurt
Texas State Bar No. 24059987
churt@bdavisfirm.com
Edward Chin (Of Counsel)
Texas State Bar No. 50511688
echin@bdavisfirm.com
Debra Coleman (Of Counsel)
Texas State Bar No. 24059595
dcoleman@bdavisfirm.com
Davis Firm, PC
213 N. Fredonia Street, Suite 230
Longview, Texas 75601
Telephone: (903) 230-9090
Facsimile: (903) 230-9661

Barry Golob
bgolob@cozen.com
Thomas Fisher
tfisher@cozen.com
Cozen O'Connor
1200 19th Street, NW
Suite 300
Washington, DC 20036
Telephone: (202) 912.4815
Facsimile: (202) 618.4843

Counsel for Plaintiff
Implicit, LLC

CERTIFICATE OF SERVICE

The undersigned certifies that the foregoing document is being filed electronically in compliance with Local Rule CV-5(a). As such, this document is being served on all counsel who are deemed to have consented to electronic service. Local Rule CV-5(a)(3)(V). Pursuant to Federal Rule of Civil Procedure 5(d) and Local Rule CV-5(d) and (e), any counsel of record not

deemed to have consented to electronic service will be served with a true and correct copy of the foregoing by email on this 28th day of January, 2020.

/s/William E. Davis, III
William E. Davis, III